UNIVERZITA PARDUBICE

FAKULTA ELEKTROTECHNIKY A INFORMATIKY

BAKALÁŘSKÁ PRÁCE

2017 Milan Suchomel

Univerzita Pardubice

Fakulta elektrotechniky a informatiky

Aplikace pro sledování geolokace

Milan Suchomel

Bakalářská práce

2017

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Akademický rok: **2016/2017**

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Milan Suchomel**
Osobní číslo:	**I14175**
Studijní program:	**B2646 Informační technologie**
Studijní obor:	**Informační technologie**
Název tématu:	**Aplikace pro sledování geolokace**
Zadávající katedra:	**Katedra informačních technologií**

Zásady pro vypracování:

Cílem bakalářské práce je vytvoření aplikace pro sledování geolokace vybraných objektů (např. osob) pro operační systém Android.

Aplikace bude disponovat dvěma režimy - sledování polohy a získání informace o poloze sledovaného objektu. Režim získání informace o poloze sledovaného objektu bude možné zpřístupnit pouze po zadání hesla.

Text bakalářské práce bude obsahovat alespoň stručný přehled a porovnání aktuálně dostupných softwarových produktů tohoto typu a analýzu a realizaci provedení aplikace.

Rozsah grafických prací:

Rozsah pracovní zprávy: **30-40 normostran**

Forma zpracování bakalářské práce: **tištěná**

Seznam odborné literatury:

1. MURACH J. Android Programming. Mike Murach & Associates, Inc. 2013. ISBN 978-1-890774-71-4.

2. ZIGURD M. Programming Android: Java Programming for the New Generation of Mobile Devices. O'Reilly Media, 2012. ISBN 978-1449316648.

Vedoucí bakalářské práce: **Ing. Michael Bažant, Ph.D.**
Katedra softwarových technologií

Datum zadání bakalářské práce: **31. října 2016**

Termín odevzdání bakalářské práce: **12. května 2017**

Univerzita Pardubice Česká republika -8-

L.S.

Ing. Zdeněk Němec, Ph.D.
děkan

Ing. Zdeněk Šilar, Ph.D.
pověřený vedením katedry

V Pardubicích dne 31. března 2017

Prohlášení autora

Prohlašuji, že jsem tuto práci vypracoval samostatně. Veškeré literární prameny a informace, které jsem v práci využil, jsou uvedeny v seznamu použité literatury.

Byl jsem seznámen s tím, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorský zákon, zejména se skutečností, že Univerzita Pardubice má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona, a s tím, že pokud dojde k užití této práce mnou nebo bude poskytnuta licence o užití jinému subjektu, je Univerzita Pardubice oprávněna ode mne požadovat přiměřený příspěvek na úhradu nákladů, které na vytvoření díla vynaložila, a to podle okolností až do jejich skutečné výše.

Souhlasím s prezenčním zpřístupněním své práce v Univerzitní knihovně.

V Pardubicích dne 9. 5. 2017

Milan Suchomel

ANOTACE

Cílem této bakalářské práce je vytvořit mobilní aplikaci pro operační systém Android, která bude sledovat geolokaci zařízení, se kterými je propojena. Poté uživatel bude mít přehled, kde se sledovaný objekt či objekty pohybují.

KLÍČOVÁ SLOVA

Android, mobilní aplikace, geolokace, GPS

TITLE

Application monitoring geolocation

ANNOTATION

The aim of this thesis is to create a mobile application for the operating system Android, which will monitor geolocation of connected devices. Then the user will have an overview of object or objects positions.

KEYWORDS

Android, mobile application, geolocation, GPS

OBSAH

SEZNAM ILUSTRACÍ A TABULEK

SEZNAM ZDROJOVÝCH KÓDŮ

SEZNAM ZKRATEK A ZNAČEK

API	Application Programming Interface
GPS	Global Positioning System
JSON	JavaScript Object Notation
PIN	Personal Identification Number
SMS	Short Message Service
SQL	Structured Query Language
Wi-Fi	Wireless Fidelity

0 ÚVOD

Cílem této bakalářské práce je realizace aplikace pro mobilní zařízení s operačním systémem Android, která umožňuje sledovat polohu zařízení pomocí GPS a jiných metod. Aplikace bude zejména zaměřena na vztah rodič-dítě, kde rodič bude moci určit místa, kde by se v daném čase mělo dítě nacházet. V závislosti na tom bude rodič informován, kde se dítě nachází a zda se pohybuje v okolí této oblasti.

První kapitola se věnuje aplikacím, které se zabývají sledováním lokace zařízení. Představeny budou aplikace, které se starají o sledování pozice druhých osob, hlavně tedy zaměřující se na vztah rodič-dítě.

Druhá kapitola se zabývá samotnou realizací již zmíněné aplikace. Nejdříve budou představeny technologie, které aplikace využívá. Následovně bude popsána struktura projektu. Nejdříve budou stručně popsány balíčky, které aplikace obsahuje. Poté budou popsány jednotlivé třídy.

V dnešní době jsou mobilní zařízení nejrozšířenější platformou. Pro tyto zařízení je vytvořeno nespočet různých aplikací pro účely komunikace, navigace nebo třeba úpravu fotografií. Existuje sice více operačních systémů, ale Android je z nich nejpoužívanější, nejen díky jeho dostupnosti. Toto odvětví se neustále rozvíjí neuvěřitelným tempem, a to je také jeden z důvodů, proč bylo vybráno téma zabývající se vývojem aplikace pro operační systém Android.

1 TEORETICKÁ ČÁST

1.1 Přehled aplikací pro sledování geolokace

V následující kapitole budou popsány aplikace, které se zabývají sledováním polohy a dalších údajů o zařízení třetích osob. Bude popsána jejich základní funkcionalita, klady a hlavně zápory těchto aplikací. Tyto aplikace jsou dostupné pro mobilní platformu Android a jsou přístupné ke stažení v Obchodě Play zdarma.

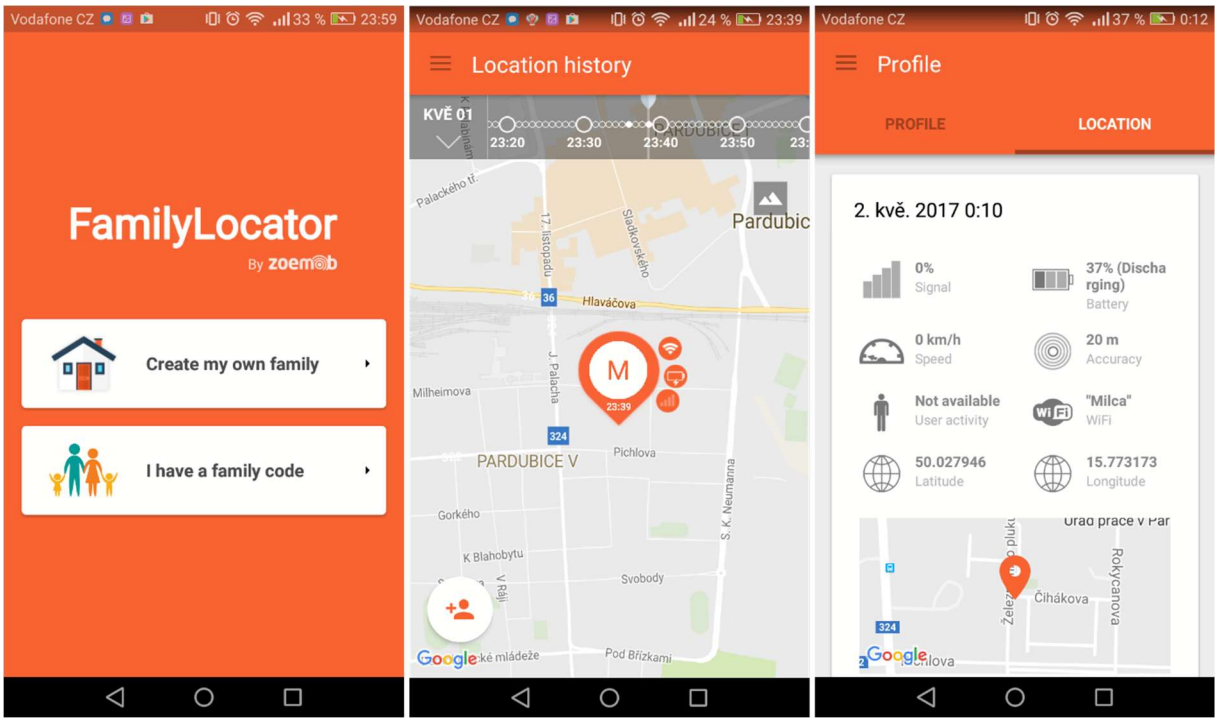
1.1.1 Family Locator

Family Locator je taková rodinná síť, kde každý člen může sdílet svoji polohu. Při prvním spuštění je uživatel vyzván k zadání telefonního čísla, poté musí zadat kontrolní kód, který mu přišel SMS zprávou. Následně má možnost vytvořit novou rodinu nebo se připojit již k vytvořené.

Po vytvoření rodiny se uživatel nachází na timeline stránce, kde má informace o členech rodiny. Další členy rodiny lze přidávat pomocí jejich telefonního čísla. Poté má uživatel přístup k informacím o jednotlivých členech rodiny. Uživatel má dále možnost posílat zprávy mezi uživateli, plánovat rodinné události a spoustu dalších možností.

Zprávy, které se automaticky posílají, obsahují spoustu informací. Mezi informace o poloze patří zeměpisná šířka a délka, přesnost změřené lokace a také rychlost pohybu. Obsahuje také informace o stavu baterie, procento nabití a zda je zařízení nabíjeno. Dále obsahuje název aktuální Wi-Fi sítě, stav mobilního signálu, uživatelovu aktivitu a hlavně čas zaznamenání těchto informací.

Aplikace je v Obchodě Play zdarma, obsahuje však premium účet, který stojí 299,99 Kč měsíčně. Poté jsou z aplikace odstraněny reklamy. Zaznamenávají se SMS zprávy a hovory. Lokace jsou uchovávány delší dobu a zruší se další omezení.



Obrázek 1 – Prostředí Family Locator

Aplikace se mi vcelku líbí, mezi zápory této aplikace patří nevalidní příchozí data, které mohou být testovaným zařízením. Signál mobilní sítě jsem nezaznamenal po celou dobu testování a uživatelskou aktivitu jsem zaznamenal hodnoty Tilting a Still, které podle mého názoru nic neříkají. Grafická stránka aplikace se mi celkem líbí, až na timeline stránku, která je podle mého mínění mírně nepřehledná.

1.1.2 Parent Control

Parent Control je aplikace na kontrolování polohy dítěte a jeho rychlosti. Pro komunikaci mezi zařízeními používá SMS komunikaci. Na první obrazovce si uživatel zvolí roli buď rodič nebo dítě, kterou nelze již změnit.

Po zvolení role rodič, je uživatel vyzván k zadání telefonního čísla dítěte. Při úspěšném propojení zařízení má rodič přístup k několika funkcím, tzn. získání aktuální lokace dítěte, dále k nastavení omezení, co se týče rychlosti a oblasti pohybu dítěte.

Uživatel v roli dítěte má přístup pouze k jedné funkci, a to poslání zprávy, která pošle lokaci a k tomu nějaké varování. Po přijetí rodičem se mu zobrazí alarm stránka a následně poloha dítěte.

Obrázek 2 – Prostředí Parent Control

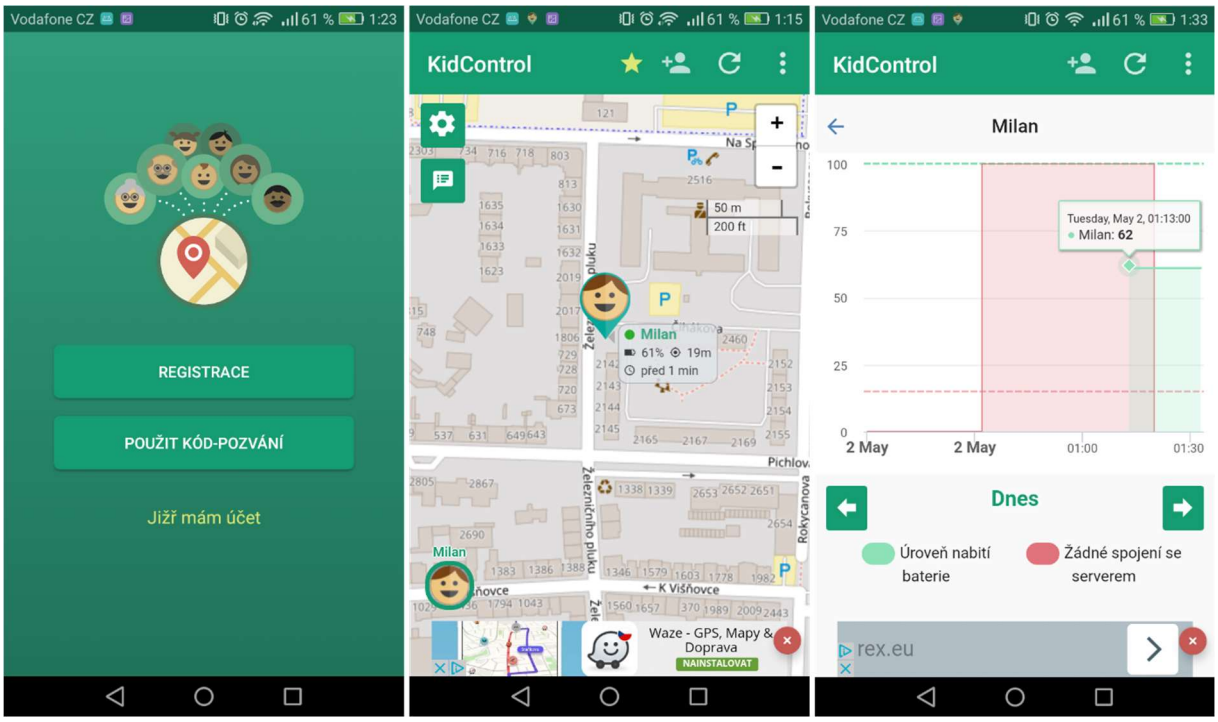
Aplikace je v Obchodě Play zdarma, ale je zastaralá a neudržovaná. Hlavní výhodou této aplikace je, že ke komunikaci používá spolehlivou mobilní síť, což je i nevýhoda, pokud účastníci nemají neomezený tarif. Tuto komunikaci bych volil pouze v případě, kdy zařízení nejsou připojena k internetu, ale komunikaci přes internet aplikace neumožňuje. Další nevýhodou je provedení aplikace po grafické stránce.

1.1.3 KidControl

KidControl je aplikace ke sledování polohy osob a stavu jejich baterie. Komunikace mezi zařízeními je zprostředkována přes internet. Aplikace opět má dva módy, rodiče a dítěte. Při prvním spuštění máme možnost registrovat se nebo připojit se pomocí kódu k již vytvořené rodině.

Po vytvoření máme možnost přidávat další členy rodiny pomocí vygenerovaného čísla. Aplikace mimo sledování informací o dětech zaznamenává informace o aktuálním uživateli. Zaznamenává základní informace o lokaci. Dále zaznamená a zobrazuje na časové ose informace o připojení k internetu a stavu baterie.

Aplikace je opět v základu zdarma, ale obsahuje možnost upgradu na premium účet jednorázovým poplatkem 169,99 Kč, který odebere z aplikace reklamy a lokace se uchovávají delší dobu.



Obrázek 3 – Prostředí KidControl

Aplikace vypadá vcelku dobře, ale obsahuje pár chyb. Jako nejzávaznější chybu vidím zobrazení značek na mapě při posunu nebo přiblížení. Značky se nahodile posouvají. Tato chyba se objevuje pouze u některých zobrazení mapy, která lze změnit v nastavení.

1.2 Shrnutí

Tabulka 1 – Porovnání aplikací

	Sledování rodiče	Podpora SMS	Hlídaná oblast	Cena	Premium [Kč]	Rodinný chat
Family Locator	ANO	NE	ANO	ZDARMA	299,9/měsíc	ANO
Parent Control	NE	ANO	ANO	ZDARMA	NE	NE
KidControl	NE	NE	NE	ZDARMA	169,9	NE
Tato aplikace	NE	Zatím NE	ANO	ZDARMA	Zatím NE	Zatím NE

2 PRAKTICKÁ ČÁST

2.1 Požadavky

Požadavky lze různě kategorizovat. Jedním z nejjednodušších je rozdělení na funkční a nefunkční. Definice požadavků jsou brány z [1].

2.1.1 Funkční požadavky

Funkční požadavky definují, co by výsledný systém měl dělat. Pro aktuální aplikaci jsem rozdělil požadavky do dvou částí, a to na funkční požadavky v roli rodiče, které jsou následovné:

- Aplikace bude umožňovat přihlášení a registraci nového uživatele.
- Aplikace bude propojovat zařízení dětí a rodičů.
- Aplikace bude přijímat a zobrazovat lokace dětí.
- Aplikace bude umožňovat vytváření oblastí, kde by se dítě mělo nacházet.

Druhou části jsou funkční požadavky na aplikaci v roli dítěte:

- Aplikace bude pravidelně zaznamenávat polohu a další informace o zařízení.
- Aplikace bude posílat data na server.

2.1.2 Nefunkční požadavky

Nefunkční požadavek je podmínka omezující daný systém. V rámci této aplikace jsou následovné:

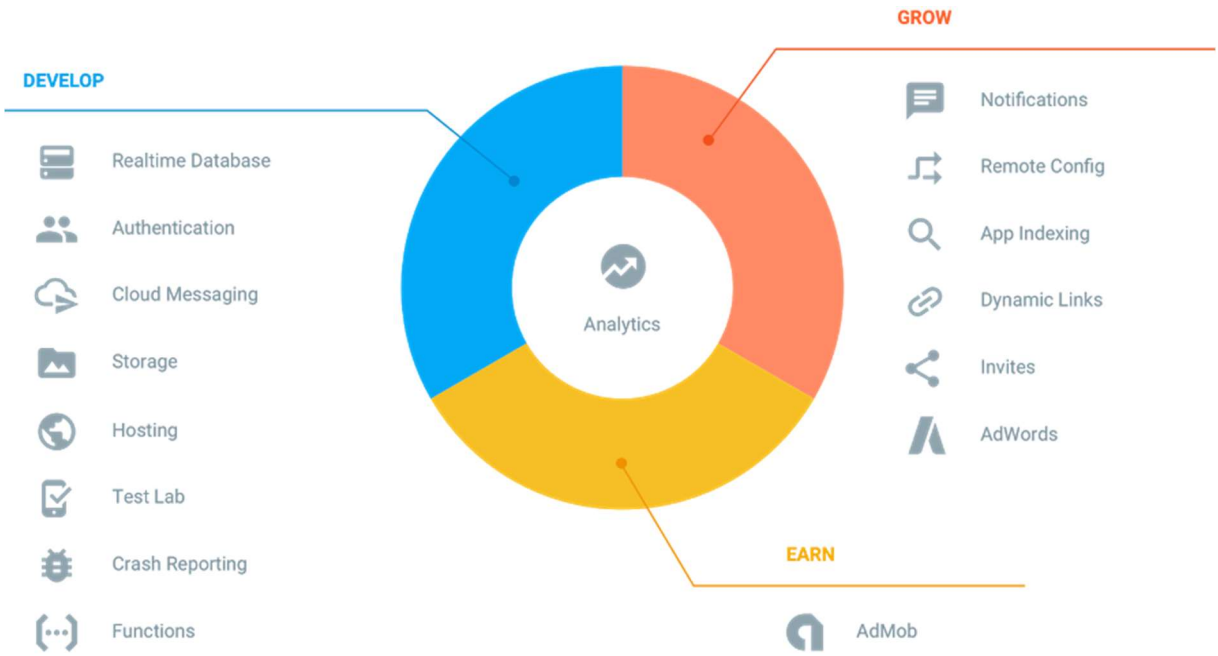
- Aplikace bude napsána v jazyce Java a v prostředí Android Studio.
- Aplikace bude komunikovat přes server Firebase.
- Aplikace bude běžet stále na pozadí.

2.2 Použité technologie

Firebase

Firebase je vývojové prostředí pro mobilní a webové aplikace. Informace o Firebase jsou čerpány z [2]. Skládá se z mnoha služeb, které může vývojář použít ve své aplikaci. Společnost byla založena v roce 2011, jejímž zakladateli byli Andrew Lee a James Tamplin. První službou byla databáze pracující v reálném čase, která poskytuje rozhraní API, které umožňuje synchronizovanou práci s daty více uživatelů. Postupem času rozšířila své služby do stávající podoby, kdy už je plnohodnotná sada pro vývoj aplikací. Společnost od roku 2014 spadá pod Google.

Většina těchto služeb je zcela zdarma jako třeba Analytics, Authentication, Cloud Messaging, Crash Reporting a další. Některé služby jako Realtime Database a Storage jsou omezeny kapacitou úložiště nebo množstvím přenesených dat. Po překročení limitů jsou k dispozici placené programy.



Obrázek 4 – Přehled služeb modulu Firebase. Zdroj: [2].

Firebase Authentication

Většina dnešních aplikací potřebuje rozlišovat jednotlivé uživatele. Znalost přihlášeného uživatele umožňuje bezpečně pracovat s daty uživatele skrze více uživatelských zařízení.

Firebase Authentication je služba pro správu uživatelů, jednoduchá na správu a implementaci do systému. Umožňuje nejen registraci a přihlášení, ale také třeba posílání e-mailů na verifikaci nebo třeba reset hesla. Uživatel se může přihlásit přes nově vytvořený účet nebo třeba přes účty Google, Facebook, Twitter nebo GitHub.

Firebase Authentication úzce spolupracuje s ostatními službami Firebase a využívá standardy, jako je OAuth 2.0 a OpenID Connect, takže lze propojit se stávajícím systémem aplikace.

Firebase Realtime Database

Firebase Realtime Database je cloudová databáze, která není založena na jazyku SQL. Umožňuje vytvářet rozsáhlé aplikace pomocí bezpečného přímého přístupu do databáze ze strany klienta. Data přetrvávají v zařízení, i když není právě připojeno k internetu. Když se

zařízení opět připojí k internetu, databáze se synchronizuje s aktuální verzí na serveru a automaticky vyřeší konflikty.

Databáze poskytuje použití flexibilního jazyka pro definování pravidel přístupů. Security Rules určují strukturu dat a pravomoci pro čtení a zápis dat. Při použití společně s Firebase Authentication lze definovat, kdo má přístup k určitým datům.

Jak již bylo zmíněno databáze je NoSQL a jako taková má různou optimalizaci a funkčnost oproti relačním databázím. Tato databáze je navržena pouze pro rychlé operace. To umožňuje vytvořit službu, která obslouží milióny uživatelů bez ztráty na rychlosti.

LocationService

LocationService se stará o získávání lokací zařízení. Této technologii bude věnována podstatná část v kapitole zabývající se třídou *BackgroundService* z balíčku *activities*.

Run time permissions

Od verze Androidu 6.0 (API level 23) jsou oprávnění přidělované během běhu aplikace, dříve byla udělena při její instalaci. Tento způsob zrychluje instalaci i aktualizace aplikace. Umožňuje tak uživateli kontrolu nad oprávněními a nemusí tak aplikaci přidělit přístup ke všem oprávněním při její instalaci. Uživatel také může kdykoliv oprávnění odebrat v nastavení aplikací.

Oprávnění se dělí na dvě základní kategorie, klasické a nebezpečné. Klasické oprávnění přímo neohrožují soukromí uživatele. Pokud tedy jsou uvedena v manifestu, jsou automaticky aplikaci přidělena. Naopak mezi nebezpečné oprávnění patří zjištění polohy nebo použití kamery. Tato oprávnění musí být přidělena za chodu aplikace a odsouhlasena uživatelem. Informace převzaty z [3].

SharedPreferences

Android poskytuje mnoho možností pro uložení dat, jednou z nich je SharedPreferences neboli sdílené nastavení. Sdílené nastavení umožňuje ukládání a získání dat formou klíčovaných hodnot do speciálního souboru. Používá se k ukládání malých dat základních datových typů. Ve třídě *Settings* bude tato technologie popsána více se způsobem použití.

SQLite

Informace o databázi SQLite jsou převzaty z [4]. Databáze SQLite je populární pro velmi malou paměťovou stopu a velmi slušnou rychlost. SQLite je velmi malou knihovnou implementující většinu ze standardu SQL-92. Přístupná je pod licencí veřejná doména, tedy zdarma pro osobní i komerční účely.

Databáze SQLite je vestavěná do běhového prostředí, může ji tedy vytvořit kterákoliv aplikace. Podrobněji je tato databáze popsána níže u popisu třídy *DBAdapterHelper*.

Toast

Informace převzaty z [4]. Informační zpráva, která se sama zobrazí a také zmizí, bez jakékoliv interakce s uživatelem. Zpráva nijak nezasahuje do rozhraní, pouze se zobrazí ve výchozím nastavení na spodní části obrazovky.

Z této zprávy nejde zjistit, zda ji uživatel zaznamenal, nečeká na potvrzení a zobrazí se pouze na krátkou dobu. Proto se používá zejména na zobrazení informace o dokončení některé události na pozadí.

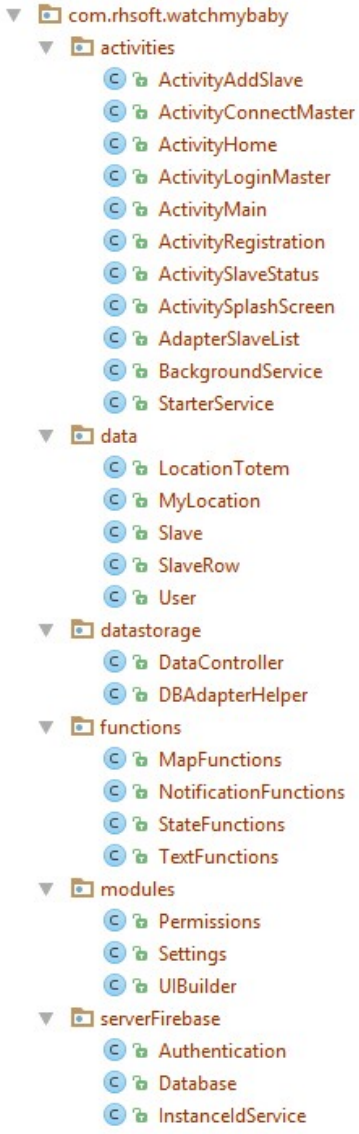
Pro vytvoření jednoduché zprávy má třída *Toast* statickou metodu *makeText()*, která vrací instanci třídy *Toast*. Požaduje aktivitu nebo jiný kontext, dále požaduje zprávu k zobrazení a hodnotu udávající životnost zobrazené zprávy. Životnost nabývá dvou hodnot *LENGTH_SHORT* a *LENGTH_LONG*.

Pro vytvoření jiných zobrazení zprávy lze vytvořit instanci třídy *Toast* pomocí konstruktoru požadující kontext. Měnit zobrazení je možné pomocí metody *setView()*. Nastavit lze také pozici, kde se má zpráva zobrazit.

Po vytvoření a nastavení zprávy ji lze zobrazit pomocí metody *show()*.

2.3 Struktura zdrojových kódů

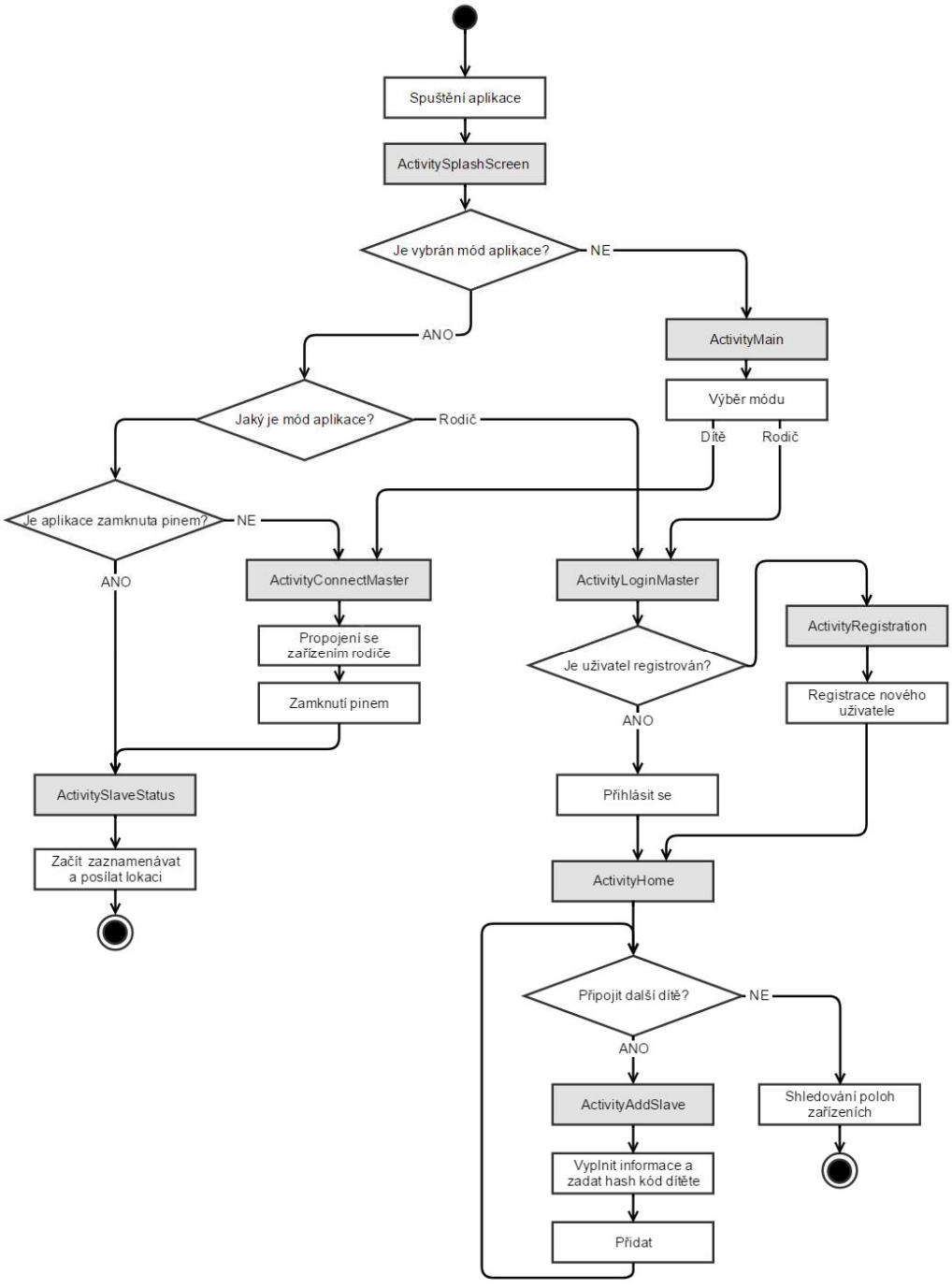
V této podkapitole bude popsána struktura zdrojového kódu a metody jednotlivých tříd. Na začátku se bude věnovat popisu jednotlivých aktivit uložených v balíčku *activities*, které zastupují zejména grafické rozvržení stránky, odpoutány od složitější logiky. Dále bude stručný popis balíčku *data*, který obsahuje pouze třídy nesoucí data. Následuje balíček *datastorage*, starající se o ukládání dat do lokální paměti. V tomto případě do databáze SQLite, která je popsána výše. Další dva balíčky *functions* a *modules* si jsou velmi podobné, obsahují třídy, ve kterých jsou pomocné metody z větší části nezávislé na aplikaci. Tyto třídy obsahují metody pro formátování času, ukládání nastavení, zobrazování dialogů a zjištění stavu zařízení. Jako poslední bude představen balíček *serverFirebase*, starající se o komunikaci se serverem. Obsahuje třídy na správu databáze na serveru a také autentifikace.



Obrázek 5 – Struktura zdrojových kódů

2.3.1 Activities

Tento balíček, jak již bylo řečeno, obsahuje jednotlivé aktivity. Nejdříve bude popsána výchozí aktivita při spuštění. Poté budou představeny dvě aktivity dostupné v módu dítě. A v neposlední řadě popsány aktivity módu rodič. Nakonec budou ukázány třídy, které pracují na pozadí. V následujícím diagramu jsou znázorněny návaznosti jednotlivých tříd.



Obrázek 6 – Diagram aktivit

ActivitySplashScreen

Pro úspěšné spuštění aplikace musí být v manifestu definována spouštěcí aktivita. Touto aktivitou je *ActivitySplashScreen*.

Tato aktivita se spustí při každém spuštění, hlavní funkcí takovéto aktivity je zobrazit se po dobu načítání aplikace. Bez této aktivity by se zobrazila pouze bílá obrazovka.

Dalším úkolem je správně uživatele přesměrovat. Při prvním, resp. při spuštění bez určeného módu aplikace bude uživatel přesměrován na aktivitu *ActivityMain* pro určení tohoto módu. Předtím však musí být mód zjištěn ze sdíleného nastavení.

Nejdříve se zjistí, zda *slaveHash* je prázdný. Pokud není prázdný, jde o mód dítěte. Poté musí zjistit, zda už je aplikace zamknuta PIN kódem, na základě toho je uživatel přesunut na aktivitu *ActivityConnectMaster* nebo *ActivitySlaveStatus*. Naopak pokud je *slaveHash* prázdný a zároveň *masterID* je větší nebo rovno nule, jde o mód rodiče a ten je přesunut na aktivitu *ActivityLoginMaster*. Toto přesměrování je vyřešeno níže v metodě *onCreate()*.

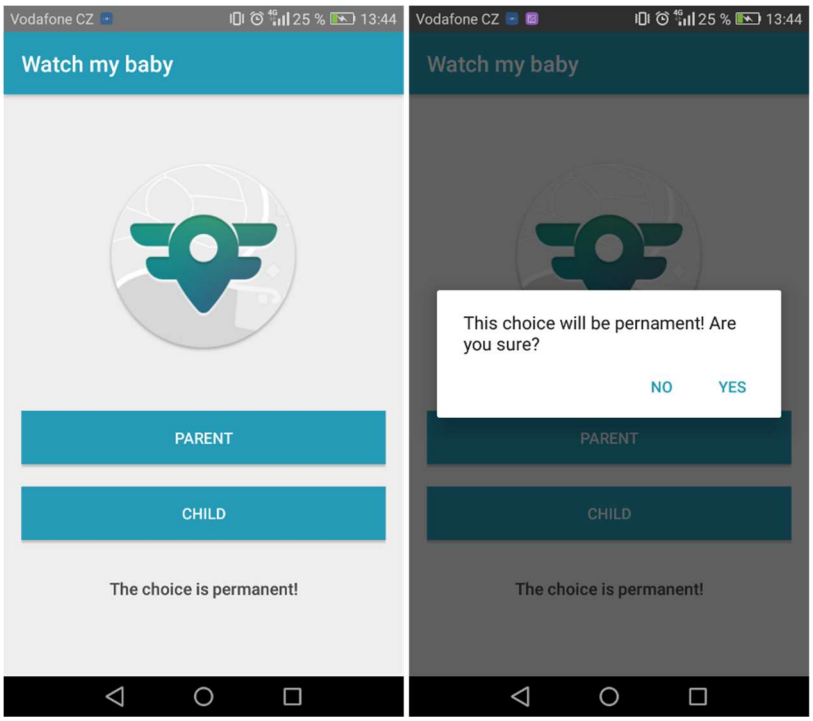
```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    Settings settings = Settings.getInstance(this);

    Class<?> activityClass = ActivityMain.class;
    if (settings.getSlaveHash() != null)
        activityClass = (settings.getPinCode() > 0) ?
ActivitySlaveStatus.class : ActivityConnectMaster.class;
    else if (settings.getMasterID() >= 0)
        activityClass = ActivityLoginMaster.class;

    startActivity(new Intent(this, activityClass));
    finish();
}
```

Kód 1 – Metoda onCreate() aktivity ActivitySplashScreen

ActivityMain

Tato aktivita má jediný úkol, a to jednoznačně určit mód aplikace. Obsahuje pouze dvě tlačítka s popisky *Parent* a *Child*. Po kliknutí na jedno z těchto tlačítek se zobrazí vyskakovací okno pro potvrzení vybraného módu. Při potvrzení je uživatel přesunut na příslušnou aktivitu.

Při výběru módu dítě je uživatel přesunut na aktivitu *ActivityConnectMaster*, která umožňuje propojení s rodičovským zařízením. Naopak po vybrání módu rodič je přemístěn na aktivitu *ActivityLoginMaster* a zároveň je hodnota *masterID* ve sdíleném nastavení na hodnotu nula.

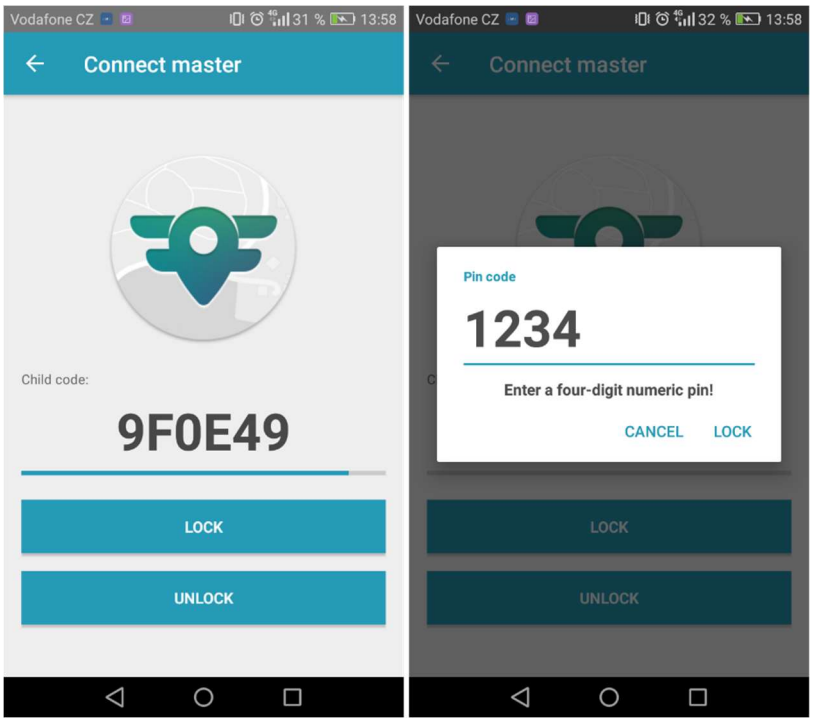


Obrázek 7 – Obrazovka ActivityMain

ActivityConnectMaster

Tato aktivita poskytuje propojení dítěte s rodiči. Nejdůležitější věcí v této aktivitě je šestimístný hash kód, který je náhodně generovaný a pro server unikátní pro jednoznačné určení zařízení, resp. dítěte. Tento kód opíše rodič do svého zařízení a tím se přihlásí k odběru polohy tohoto zařízení. Pod tímto kódem se nachází tlačítko pro zamknutí a odemknutí. Kód je přístupný na propojení pouze po určitou dobu, kterou lze prodloužit nebo vynulovat pomocí tlačítek. Dobu, kdy je kód aktivní zobrazuje ukazatel pod hash kódem.

Po kliknutí na tlačítko pro zamknutí je uživatel vyzván k zadání PIN kódu. Tento kód je vyžadován při pokusu dostat se zpět na tuto aktivitu. Po zadání čtyřmístného číselného kódu je uživatel přesunut na aktivitu *ActivitySlaveStatus*.

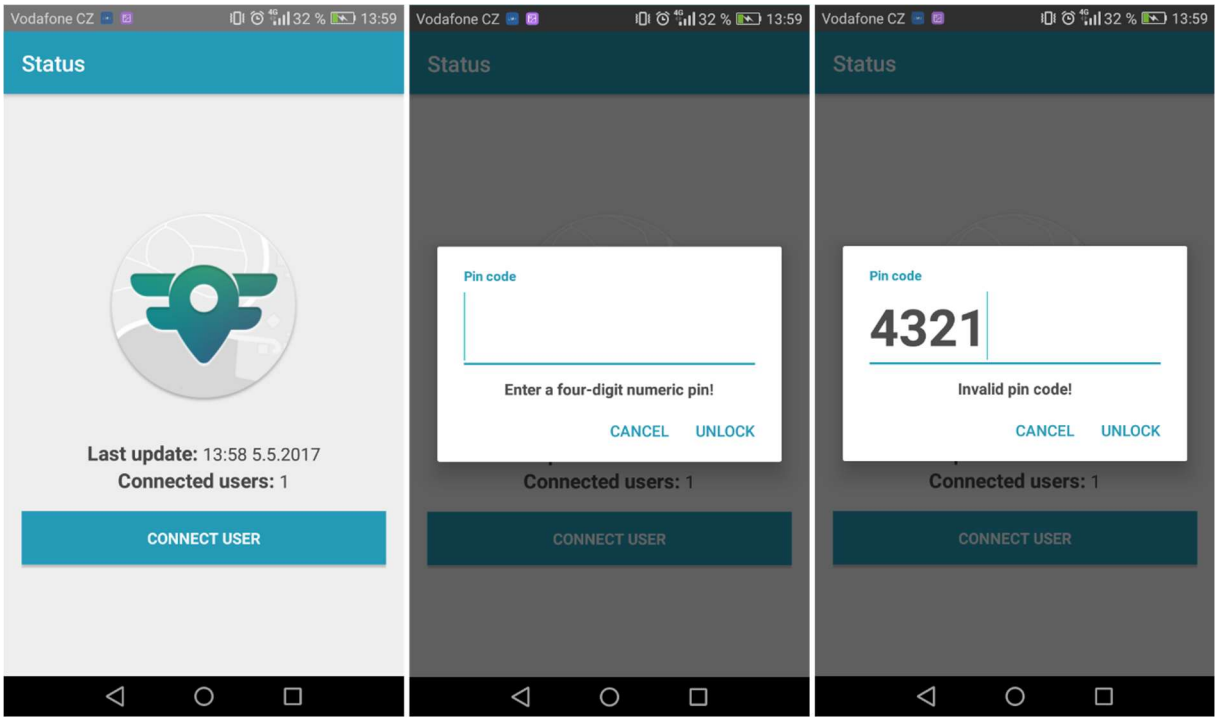


Obrázek 8 – Obrazovka ActivityConnectMaster

ActivitySlaveStatus

Výchozí obrazovka při spuštění aplikace v roli dítěte. Při vytvoření této aktivity se zároveň spustí servisní služba na pozadí *BackgroundService*, popsána na konci této podkapitoly. Tato aktivita zobrazuje pouze omezené informace, a to čas posledního zaznamenání lokace, dalších informací a počet připojených zařízení k odběru těchto informací.

Tato aktivita také obsahuje tlačítko, které uživateli umožní přesun na aktivitu *ActivityConnectMaster*. Předtím je však uživatel vyzván k zadání dříve zadaného PIN kódu.

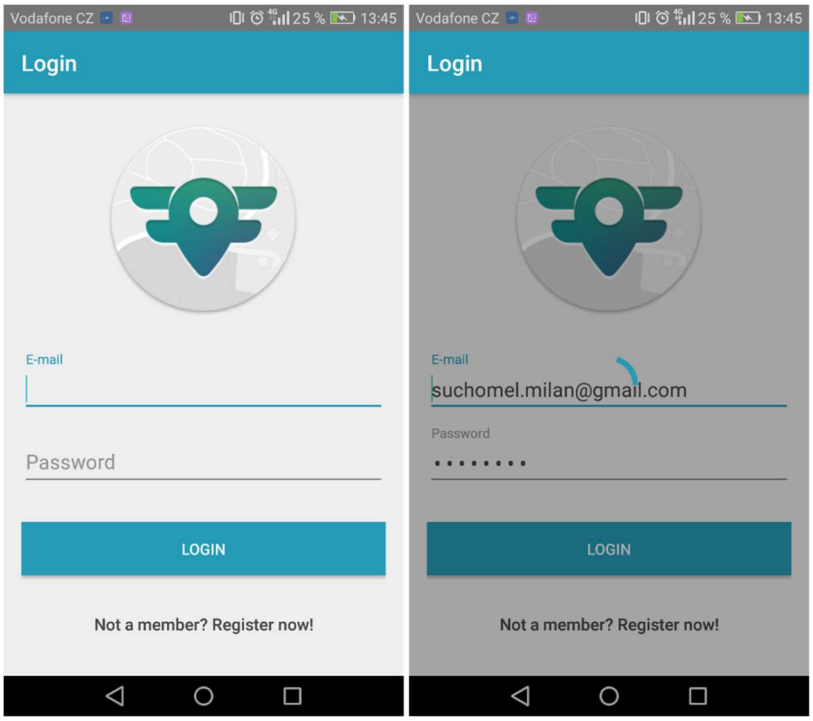


Obrázek 9 – Obrazovka ActivitySlaveStatus

ActivityLoginMaster

Tato aktivita, jak již název napovídá, zajišťuje přihlášení uživatele do aplikace. Ze všeho nejdříve zkontroluje, jestli uživatel není stále přihlášen. Zkontroluje tedy hodnoty *email* a *password* ze sdíleného nastavení a pokud nejsou prázdné tak se pokusí s těmito údaji přihlásit.

Pokud tomu tak není, uživatel má možnost přihlášení. Po vyplnění přihlašovacího emailu, hesla a stisknutí tlačítka pro přihlášení, se zavolá metoda *loginUser()* volaná nad instancí třídy *Authentication*. Tato metoda zajistí přihlášení do systému. Při úspěšném přihlášení je uživatel přesunut na aktivitu *ActivityHome*. Naopak při chybě je uživatel informován chybovou hláškou. Více o této metodě je v podkapitole níže, která se věnuje balíčku *serverFirebase*.



Obrázek 10 – Obrazovka ActivityLoginMaster

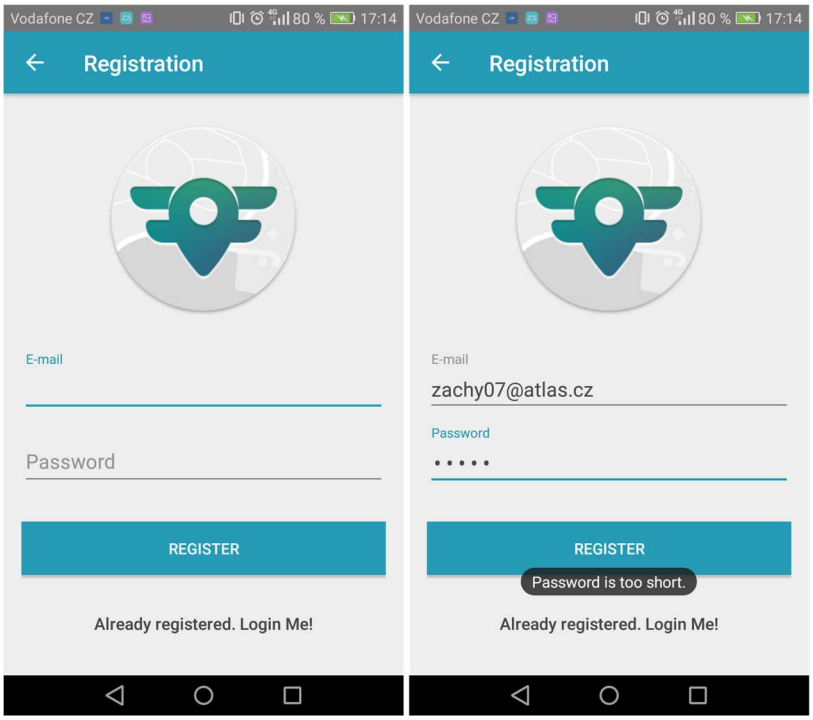
Pro nové uživatele je možnost registrace uživatele pod tlačítkem pro přihlášení. Po stisknutí je uživatel přesunut na aktivitu *ActivityRegisterMaster*.

ActivityRegisterMaster

Tato aktivita se nejen vizuálně velmi podobá aktivitě *ActivityLoginMaster*, umožňuje však registraci nových uživatelů. Obsahuje pouze dvě povinná pole:

- Email – použit jako přihlašovací jméno, musí být ve validním email formátu.
- Heslo – delší než 6 znaků.

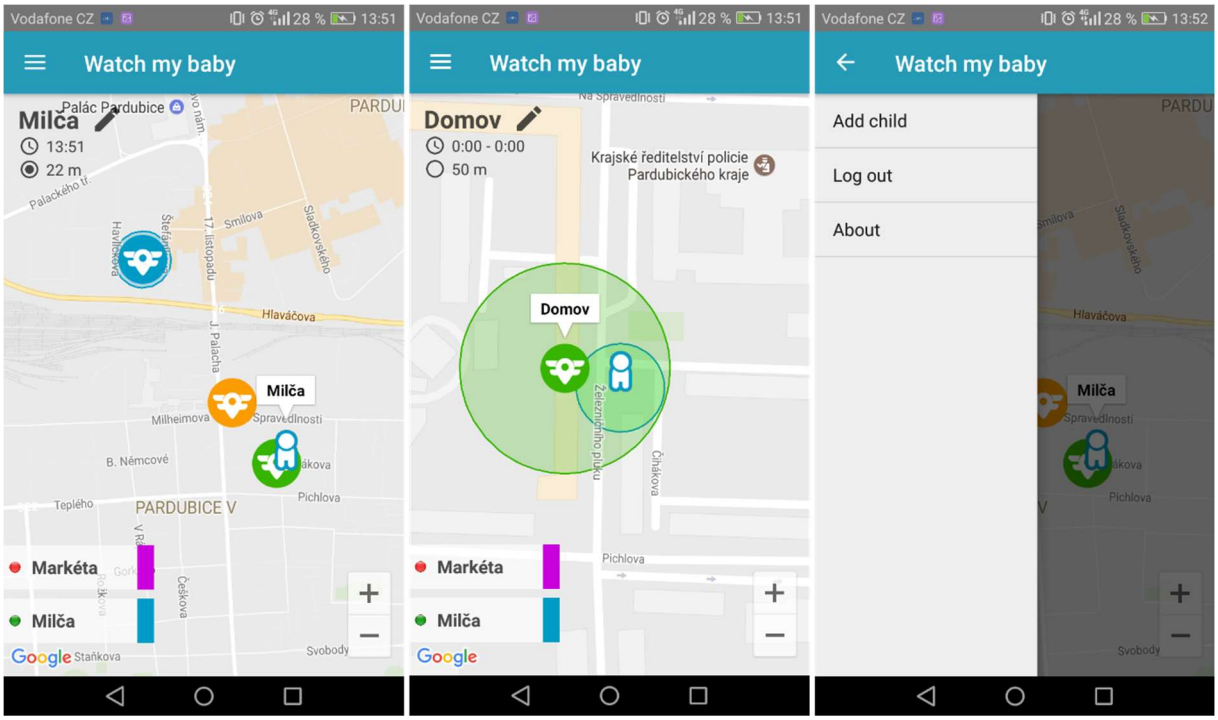
Po kliknutí na tlačítko pro registraci jsou zkontrolována obě pole, zda nejsou prázdná. Následně je zavolána metoda *registerUser()*, volaná nad instancí třídy *Authentication*. Tato funkce jako funkce *loginUser()* je popsána níže v podkapitole *firebase*. Po úspěšné registraci je uživatel automaticky přihlášen a přesunut na aktivitu *ActivityHome*. Při neúspěšné registraci je chyba vypsána pomocí *Toast* na spodní části obrazovky.



Obrázek 11 – Obrazovka ActivityRegisterMaster

ActivityHome

Hlavní a také výchozí aktivita pro přihlášeného uživatele. V této aktivitě je hlavním prvkem Google mapa, která zobrazuje poslední obdržené lokace dětí a všechny totemy tedy oblasti, kde by se děti měly nalézat. Dále obsahuje v levé dolní části seznam dětí. Nahoře jsou vidět informace o vybraném objektu. Součástí této aktivity je boční menu. Všechny tyto součásti jsou popsány v dalších odstavcích.



Obrázek 12 – Obrazovka ActivityHome

Mapa zobrazuje aktuální lokace dětí a aktualizuje se, jakmile přijde další lokace od některého z dětí. Lokace dítěte se zobrazuje jako značka s vybranou ikonou ohraničená kruhem, který reprezentuje přesnost naměřené lokace, tedy v jakém okruhu se dítě nalézá. Tento okruh má barvu podle vzdálenosti od aktivních totemů. Tyto stavy jsou reprezentovány následujícími barvami:

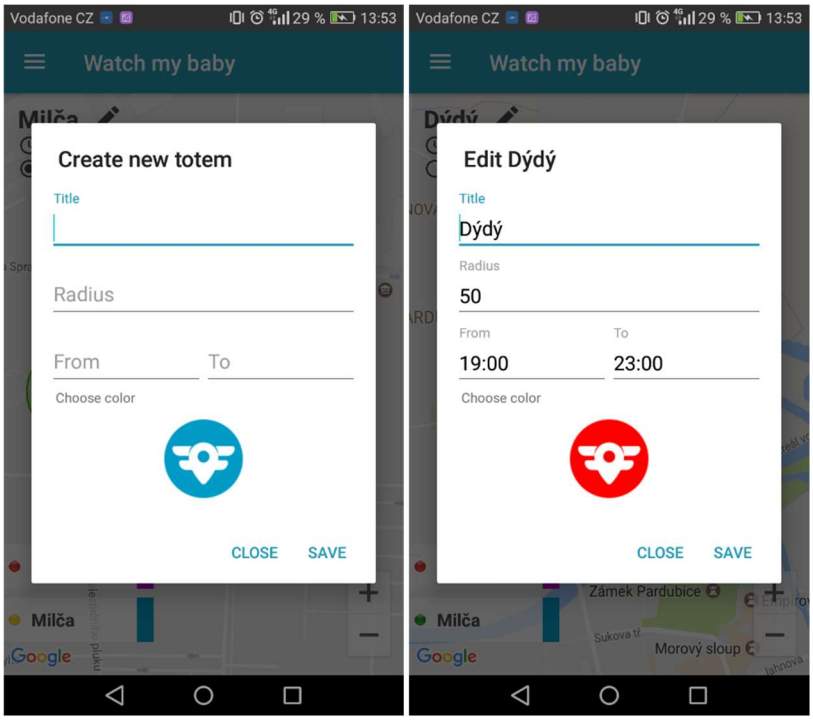
- Zelená – dítě se vyskytuje u některého z totemů.
- Oranžová – dítě může a nemusí být u některého z totemů, resp. je na hranici totemu.
- Červená – dítě není u žádného z aktivních totemů.
- Šedá – v tomto čase není žádný totem aktivní.

Po kliknutí na ikonu dítěte se objeví jeho jméno a zároveň se zobrazí informace o lokaci v levém horním rohu. Tyto informace zobrazují čas zaznamenání, přesnost v metrech

naměřené lokace a rychlost, pokud byla zaznamenána. Navíc je tam tlačítko s ikonou tužky pro editaci aktuálního dítěte, ta po kliknutí uživatele přesune na aktivitu *ActivityAddSlave*.

Totemy jsou zobrazovány jako značky s okruhem, který symbolizuje poloměr daného totemu. Po kliknutí na tuto značku se opět zobrazí informace. Tyto informace obsahují časy, kdy je totem aktivní a poloměr oblasti. Také obsahuje tlačítko na editaci, které otevře vyskakovací okno. Pokud uživatel nemá totem ještě vytvořený, může ho vytvořit kliknutím na mapu. Po dlouhém stisknutí totemu může být přesunut.

Obrázek níže ukazuje vyskakovací okno pro vytváření a editaci totemu. O totemu lze nastavit pár informací, a to název oblasti, poloměr oblasti, barvu a časy, kdy je totem aktivní. Barvu lze vybrat po kliku na barevnou ikonu totemu. Po kliknutí se uživateli zobrazí paleta předem definovaných barev, ze kterých si uživatel vybere. Po potvrzení se totem zapíše do databáze Firebase.



Obrázek 13 – Okno pro vytvoření a editaci totemu

V levé dolní části se nalézá list dětí, který zobrazuje stav, jméno a barevné označení dítěte. Po kliku na jedno z dětí se mapa plynule přesune k poslední lokaci tohoto dítěte.

Poslední součástí této aktivity je boční menu, které umožňuje uživateli provést odhlášení, tedy vymazat údaje ze sdíleného nastavení a přemístění na stránku pro přihlášení. Dále umožňuje propojit se s dalšími zařízeními pomocí aktivity *ActivityAddSlave*.

ActivityAddSlave

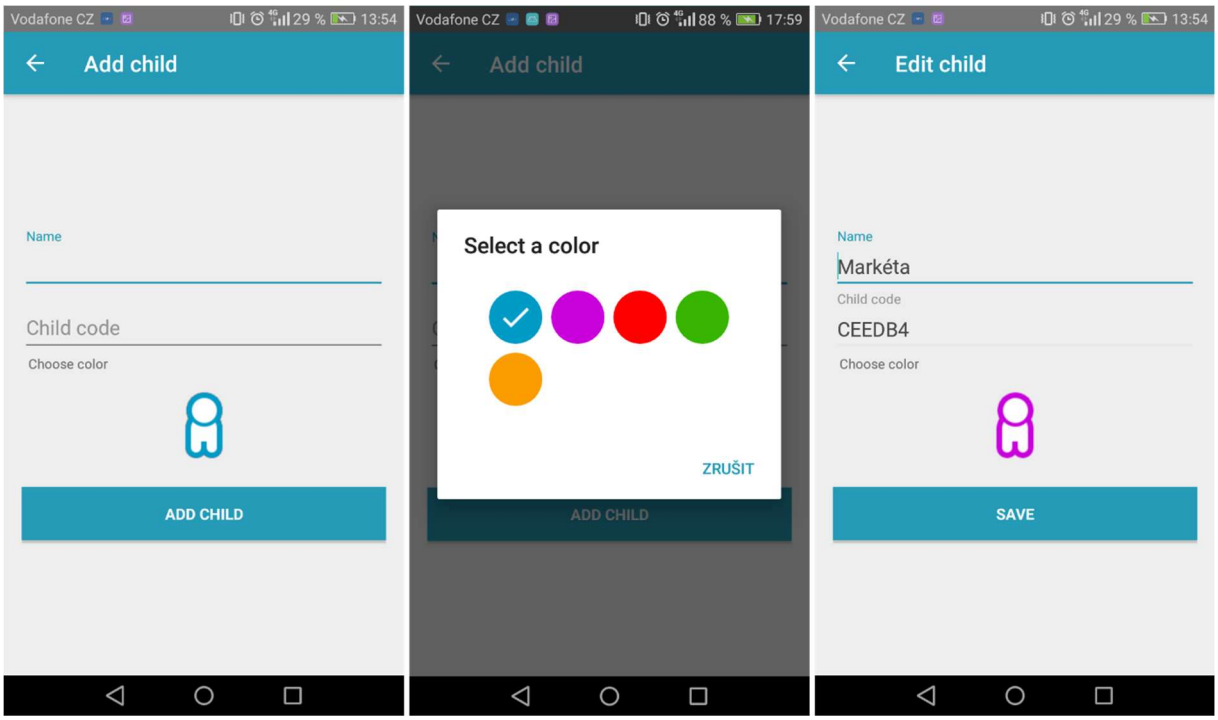
Tato aktivita slouží pro připojení dalších zařízení ke sledování. K úspěšnému propojení je nutné vyplnit následující pole:

- Name – jméno nebo přezdívka dítěte.
- Child code – šestimístný hash, jednoznačně reprezentující jiné zařízení.

Uživatel si také může vybrat barvu ikony, která bude představovat dítě na mapě. Pro změnu této barvy je nutné kliknout na barevnou ikonu dítěte. Poté se uživateli zobrazí vyskakovací okno pro výběr barvy. Uživatel má možnost si vybrat jednu z pěti barev. Po potvrzení se automaticky zobrazí příslušná ikona.

Po vyplnění těchto hodnot je ve spodní části tlačítko pro přidání dítěte. Toto tlačítko přesune uživatele zpět na aktivitu *ActivityHome* a hlavně připojí uživatele k odběru lokací nově připojeného zařízení. Což je podrobně popsáno níže v podkapitole zabývající se modulem Firebase Realtime Database.

Tato aktivita je použita také pro úpravu již připojených dětí. Při vytváření aktivity se předá *slaveID*, vyplní se všechna pole a zobrazí se příslušná ikona. Pole *child code* již není povoleno k dalším úpravám. Je také změněn titulek a popis u tlačítka na *Edit child*, jak je vidět v obrázku níže.



Obrázek 14 – Obrazovka ActivityAddChild

BackgroundService

Je služba, která pracuje na pozadí. Tato třída dědí ze třídy *IntentService* a implementuje několik rozhraní. Tato služba běží pouze v módu dítěte a zajišťuje zaznamenání a nahrávání dat na server. Pro zaznamenávání polohy je použita třída *LocationService*, a proto je nutné vytvořit instanci třídy *GoogleApiClient*, hlavní přístupový bod do služeb Googlu.

Mezitím co se klient připojuje, může být nastaven *LocationRequest*. Nastavuje se zde interval aktualizování lokace a také jakou prioritu má její zaznamenání. Priorita má dva extrémy. Prvním je, kdy aplikace sama o sobě nevyžaduje sbírání lokací, ale získá lokaci pokaždé, když si ji vyžádá některá jiná. Prioritu je pak nutné nastavit na hodnotu finální proměnné *PRIORITY_NO_POWER*. V této aplikaci však je přesnost a frekvence zaznamenání lokace velmi důležitá, a proto je zvolen druhý extrém, a to nastavení častých a přesných aktualizací lokací pomocí priority *PRIORITY_HIGH_ACCURACY*. S tímto přímo souvisí intervaly aktualizací. Prvním je interval, kdy je vyžádána aktualizace aplikací, ten je nastaven na hodnotu deset tisíc milisekund tedy deset sekund. Druhým je tzv. nejrychlejší interval, ten získává lokace z cizích aplikací a je nastaven na pět sekund.

```
private void setupLocationServices() {
    mLocationRequest = LocationRequest.create();
    mLocationRequest.setPriority(LocationRequest.PRIORITY_HIGH_ACCURACY);
    mLocationRequest.setInterval(10 * 1000);
    mLocationRequest.setFastestInterval(5 * 1000);

    LocationSettingsRequest.Builder builder = new
LocationSettingsRequest.Builder()
            .addLocationRequest(mLocationRequest);

    builder.setAlwaysShow(true);
}
```

Kód 2 – Metoda nastavující LocationRequest

Při vytváření Google klienta byla také nastavena metoda, která se zavolá po úspěšném připojení tohoto klienta. Po úspěšném připojení lze požadovat lokace zařízení, proto se zavolá metoda *startLocationUpdate()*. Nejdříve tato metoda zkontroluje, zda má aplikace oprávnění k získávání poloh, a poté se přihlásí k odběru lokací, funkcí *requestLocationUpdates()* požadující jako parametry instance tříd *GoogleApiClient*, *LocationRequest* a také instanci třídy, která implementuje rozhraní *LocationListener*. Toto rozhraní přidává metodu *onLocationChange()*, která obsahuje parametr *Location* udržující všechny informace o získané lokaci. Tato metoda se zavolá při každé aktualizaci lokace.

StarterService

Poslední třídou tohoto balíčku je třída zajišťující spuštění služby BackgroundService po restartu aplikace. Pro tyto účely musí být v manifestu oprávnění na získání zprávy po restartu *RECEIVE_BOOT_COMPLETED*. A také tato třída musí být zapsána v manifestu, jak je tomu níže.

```
<receiver android:name=".activities.StarterService">
    <intent-filter>
        <action android:name="android.intent.action.QUICKBOOT_POWERON" />
        <action android:name="android.intent.action.BOOT_COMPLETED" />
    </intent-filter>
</receiver>
```

Kód 3 – Definice StarterService v manifestu

2.3.2 Data

Balíček data obsahuje třídy, které nesou data v proměnných a obsahují metody *get()* a *set()* k jejich získání. Obsahuje následující třídy:

- LocationTotem – uchovává informace o totemech tedy oblastech, kde se dítě má nacházet. Například lokaci, poloměr nebo aktivní dobu oblasti.
- MyLocation – uchovává zeměpisnou délku, zeměpisnou šířku, přesnost lokace, rychlost pohybu a další informace o poloze.
- Slave – udržuje informace o připojených dětech jako jméno, barvu a další.
- SlaveRow – obsahuje některé informace o dítěti rozšířené o poslední známou lokaci a aktivní totemy. Tato třída se používá v adaptéru *AdapterSlaveList*.
- User – uchovává informace o přihlášeném uživateli jako přihlašovací email, jméno, příjmení a další.

Některé ze tříd obsahují metodu *getMap()* pro vytvoření mapy z objektu. Tato mapa obsahuje klíčované hodnoty určené k nahrání na server. V obrázku níže je ukázána metoda *getMap()* ze třídy *MyLocation*.

```
@Exclude
public Map<String, Object> toMap() {
    HashMap<String, Object> result = new HashMap<>();
    result.put(EMAIL, email);
    result.put(FORENAME, forename);
    result.put(SURNAME, surname);
    result.put(CREATED, created);
    result.put(MODIFIED, modified);
    result.put(CLOUD_MESSAGING_TOKEN, (new InstanceIdService()).getToken());

    return result;
}
```

Kód 4 – Metoda toMap() ve třídě User

2.3.3 DataStorage

Balíček *dataStorage* se stará o ukládání lokálních dat do SQLite databáze. Obsahuje dvě třídy *DataController* a *DBAdapterHelper*. Obě třídy jsou postaveny na návrhovém vzoru jedináček, tedy je možnost vytvoření pouze jedné instance třídy. A nakonec bude popsán relační model této databáze.

Relační model

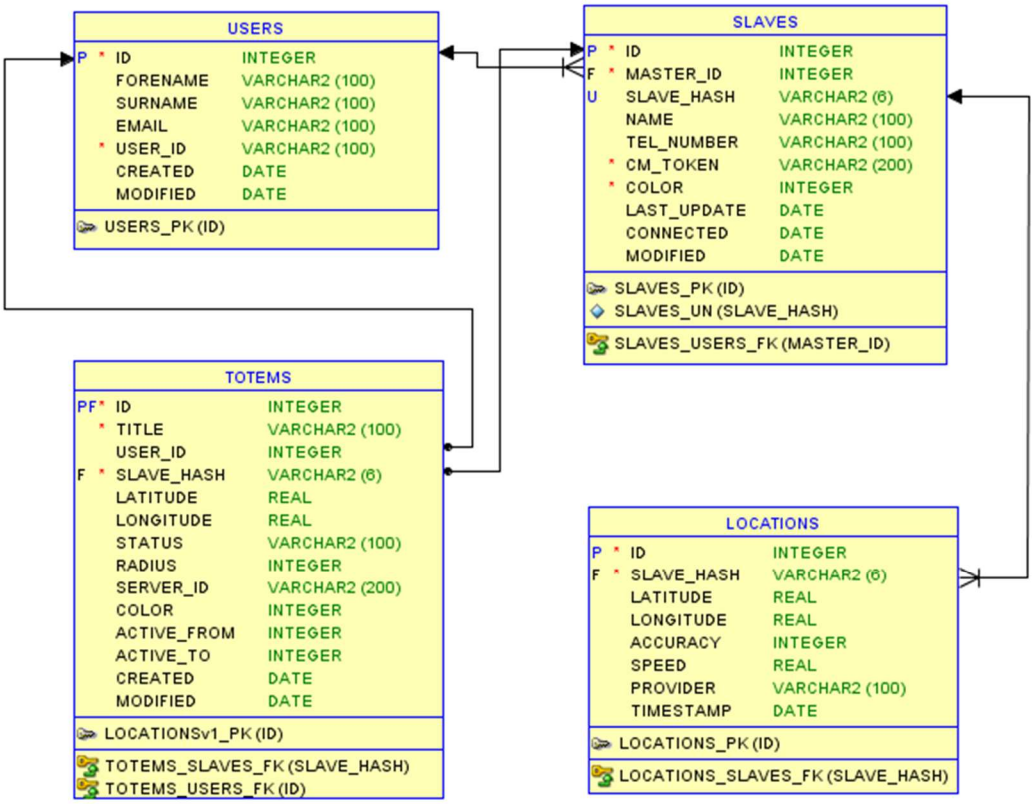
Relační model zobrazuje logický pohled na hierarchickou strukturu databáze. Je tvořen tabulkami a relacemi mezi nimi. V následujících odstavcích budou stručně popsány tabulky a nastíněny vazby mezi nimi.

Databáze se jménem *WATCH_MY_BABY* se skládá pouze ze čtyř tabulek. Tyto tabulky a celá databáze je používána pouze v roli rodiče. Obsahuje tabulky pro ukládání informací o přihlášených uživatelích, dětech, se kterými jsou tito uživatelé propojeni. Dále tabulku obsahující záznamy lokací, které se stáhly ze serveru a tabulku totemů.

Tabulka *USERS* tedy obsahuje informace o uživateli jako jméno, email, časy modifikací a také serverem vytvořený unikátní klíč. V tabulce *SLAVES* jsou uloženy informace o dětech jako pro server unikátní hash kód, jméno, čas propojení s uživatelem a samozřejmě identifikační číslo uživatele se kterým se propojil. Dále se uchovávají informace o stažených lokacích, a to do tabulky *LOCATIONS*. V této tabulce jsou informace o lokaci. Mezi ty důležité informace patří zeměpisná šířka, zeměpisná délka, čas zaznamenání a samozřejmě unikátní hash kód dítěte pro úspěšnou identifikaci. Poslední tabulkou v modelu je tabulka pro

udržování totemů, tedy vymezených lokací pro dítě, která se jmenuje *TOTEMS*. Tato tabulka je největší, co se týče počtu sloupců. Obsahuje informace o pozici, kde se tato oblast nachází nebo poloměr oblasti. Dále také časy od kdy do kdy je totem aktivní. A třeba také jakou barvou se má zobrazovat na mapě.

Obrázek níže zobrazuje již zmíněný relační model, jeho tabulky a také jednotlivé vazby mezi nimi. Jsou v něm znázorněny také jednotlivé sloupce tabulky s jejich datovými typy. Pod výpisem sloupců jsou vypsána omezení jako primární a cizí klíče. Tento model je tvořen v programu SQL Developer.



Obrázek 15 – Relační model databáze SQLite

DataController

Tato třída je pouhou mezivrstvou mezi třídou *DBAdapterHelper* a zbytkem aplikace. Ostatní třídy nemají přístup k metodám této třídy.

DBAdapterHelper

Třída *DBAdapterHelper* dědí ze třídy *SQLiteOpenHelper*, která zajišťuje vytvoření, verzování a práci s databázovým souborem. Při vytvoření instance této třídy se zjistí, zda tento soubor existuje, pokud ne, zavolá se metoda *onCreate()*, která definuje tabulky. Při existenci tohoto souboru se zkontroluje verze a pokud se liší, tak se zavolá metoda *onUpgrade()*. V této metodě definujeme změny databáze, které tabulky se upravily nebo třeba smazaly. Do této metody vstupuje stávající SQLite databáze a číslo staré a nové verze.

```
private static DBAdapterHelper db;

private DBAdapterHelper(Context ctx) {
    super(ctx, DATABASE_NAME, null, DATABASE_VERSION);
}

protected static synchronized DBAdapterHelper getInstance(Context ctx) {
    if (db == null)
        db = new DBAdapterHelper(ctx);
    return db;
}

@Override
public void onCreate(SQLiteDatabase db) {
    db.execSQL(CREATE_TABLE_USERS);
    db.execSQL(CREATE_TABLE_LOCATIONS);
    db.execSQL(CREATE_TABLE_TOTEMS);
    db.execSQL(CREATE_TABLE_SLAVES);
}

@Override
public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion)
{

}
```

Kód 5 – Vytvoření SQLite databáze

Pod finálními proměnnými začínající řetězcem *CREATE_TABLE* se skrývají SQL dotazy na vytvoření jednotlivých tabulek. V obrázku níže je SQL dotaz na vytvoření tabulky obsahující uživatele, kteří aplikaci používají.

```
private static final String CREATE_TABLE_USERS =
        "create table if not exists "
        + TABLENAME_USERS + "("
        + ID + " integer primary key,"
        + FORENAME + " text,"
        + SURNAME + " text,"
        + EMAIL + " text,"
        + USER_ID + " text,"
        + CREATED + " DATETIME DEFAULT CURRENT_TIMESTAMP,"
        + MODIFIED + " DATETIME DEFAULT CURRENT_TIMESTAMP);";
```

Kód 6 – Vytvoření tabulky uživatelů

Dále tato třída obsahuje metody, pomocí kterých se vkládají, aktualizují nebo vybírají data z jednotlivých tabulek. Po získání databáze pomocí metody *getWritableDatabase()* nebo *getReadableDatabase()*, podle typu použití, lze zpracovat dotaz dvěma způsoby. První je, že se do metody *execSQL()* předá klasický SQL dotaz. Druhým způsobem je předání příslušné metodě požadovaná data, tento způsob je popsán v dalších odstavcích.

Pro vkládání dat je tu metoda *insert()*. Tato metoda vyžaduje název tabulky a hodnoty, které se mají do tabulky vložit. Tyto hodnoty jsou v podobě klíčovaných hodnot uloženy v instanci třídy *ContentValues*. Návratovou hodnotou této metody je identifikační číslo, pod kterým se data uložila. Další zapisovací metodou je metoda *update(),* která upraví již vytvořená data. Mimo výše zmíněné, je možné vyplnit klauzuli *where* určující, nad kterými řádky tabulky se mají změny provést. Návratovým typem této metody je počet upravených řádků. Ukázka těchto metod je ukázána na vkládání nových uživatelů do lokální databáze.

```
void upsertUser(User user) {
    ContentValues vals = new ContentValues();
    vals.put(SURNAME, user.getSurname());
    vals.put(FORENAME, user.getForename());
    vals.put(EMAIL, user.getEmail());
    vals.put(CREATED, user.getCreated());
    vals.put(MODIFIED, user.getModified());
    vals.put(USER_ID, user.getUserID());

    int updates = db.getWritableDatabase().update(TABLENAME_USERS, vals,
EMAIL + "='" + user.getEmail() + "'", null);
    if (updates == 0)
        user.setDbID(db.getWritableDatabase().insert(TABLENAME_USERS,
null, vals));
}
```

Kód 7 – Vložení uživatelů do SQLite

Metodou pro čtení dat je metoda *query()*. Tato metoda opět požaduje název tabulky a také různé klauzule jako where, group by, having nebo třeba limit. Je možné také určit, které sloupce se mají vybrat. Tato metoda vrací kurzor, který obsahuje všechna přečtená data. Tento kurzor je nutné po řádcích projít a poskládat z nich objekty. Příklad této metody je ukázán níže na čtení z tabulky uživatelů.

```
private List<User> selectUsers(String whereClause) {
    Cursor c = db.getReadableDatabase().query(TABLENAME_USERS, null,
whereClause, null, null, null, null, null);
    c.moveToFirst();
    List<User> userList = new ArrayList<User>();

    while (!c.isAfterLast()) {
        User user = new User();
        user.setDbID(c.getLong(c.getColumnIndex(ID)));
        user.setCreated(c.getLong(c.getColumnIndex(CREATED)));
        user.setModified(c.getLong(c.getColumnIndex(MODIFIED)));
        user.setEmail(c.getString(c.getColumnIndex(EMAIL)));
        user.setSurname(c.getString(c.getColumnIndex(SURNAME)));
        user.setForename(c.getString(c.getColumnIndex(FORENAME)));
        user.setUserID(c.getString(c.getColumnIndex(USER_ID)));

        userList.add(user);
        c.moveToNext();
    }
    c.close();
    return userList;
}
```

Kód 8 – Získání uživatelů z SQLite

2.3.4 Functions

Tento balíček obsahuje třídy s pomocnými metodami, všechny metody v těchto třídách jsou statické. Třídám *MapFunctions* a *TextFunction* budou věnovány následující podkapitoly. Dále také obsahuje třídu *NotificationFunctions*, která se stará o zobrazování zpráv.

MapFunction

Tato třída se stará o práci s mapou. Zobrazuje lokace dětí na mapě. Stará se o přesun kamery k zaměřené lokaci a výpočet vzdálenosti mezi dvěma body na mapě.

Vykreslení lokace na mapu se skládá ze dvou metod, nejprve se lokace připraví a následně se spočítá její stav k aktivním totemům. Připraví se ikona a vykreslený okruh. Toto se připraví v separátním vlákně v metodě *prepareSlaveLastLocation()*. Následně je volaná metoda z hlavního vlákna *showSlaveLastLocation()*, ve které se na mapu tyto připravené informace vykreslí.

Metoda pro vypočítání vzdáleností mezi body zadanými v zeměpisné šířce a zeměpisné délce je ukázána v textovém poli níže.

```
public static double calculateDistance(double lat1, double lng1, double
lat2, double lng2) {
    double earthRadius = 6371000;
    double dLat = Math.toRadians(lat2-lat1);
    double dLng = Math.toRadians(lng2-lng1);
    double a = Math.sin(dLat/2) * Math.sin(dLat/2) +
            Math.cos(Math.toRadians(lat1)) *
Math.cos(Math.toRadians(lat2)) *
                    Math.sin(dLng/2) * Math.sin(dLng/2);
    double c = 2 * Math.atan2(Math.sqrt(a), Math.sqrt(1-a));
    return earthRadius * c;
}
```

Kód 9 – Metoda na výpočet vzdálenosti

TextFunctions

Tato třída obsahuje metody starající se hlavně o formát času, ale také o změnu velikosti obrázku nebo získání barvy podle názvu. Název třídy není příliš vhodně zvolen.

Čas udaný v milisekundách lze formátovat pomocí třídy *SimpleDateFormat,* které se v konstruktoru předá předpis, v jakém se má čas vypsat. Následně se nad ní zavolá metoda *format()* s instancí třídy *Date*. Tato metoda je ukázána v kódu níže.

```
public static String getTime(long time) {
    Date date = new Date(time);
    DateFormat formatter = new SimpleDateFormat("HH:mm:ss - dd.MM.yy");
    return formatter.format(date);
}
```

2.3.5 Modules

Permissions

Tato třída se stará o vyžadování oprávnění, která potřebují potvrzení uživatelem, na systémech Android s verzí 6.0 a vyšší. V této aplikaci je vyžadováno pouze jedno oprávnění tohoto typu, a to pro získávání lokace zařízení *ACCESS_FINE_LOCATION*. Obsahuje metodu *requestPermission()*, která zobrazí vyskakovací okno, pokud již není oprávnění povoleno. Toto okno se nedá jakýmkoliv způsobem upravit, aby uživatel věděl, co potvrzuje.

Settings

Třída ukládající základní nastavení a informace o přihlášeném uživateli. K ukládání využívá sdílené nastavení zmíněné v kapitole použité technologie. Umožňuje ukládání celých čísel,

desetinných čísel, textových řetězců a množin textových řetězců. Hodnota má vždy přiřazen klíč, pod kterým jsou hodnoty ukládány, jedná se tedy o klíčované hodnoty. V ukázce níže vidíme vkládání a čtení textového řetězce do sdíleného nastavení. Sdílené v tomto smyslu znamená, že jsou přístupné ze všech aktivit, nejen z jedné.

```
public void setEmail(String email){
    setPreference(EMAIL, email);
}

private void setPreference(String key, String value) {
    SharedPreferences settings = ctx.getSharedPreferences(PREFS_NAME,
MODE_PRIVATE);
    SharedPreferences.Editor editor = settings.edit();
    editor.putString(key, value);
    editor.apply();
}

public String getEmail(){
    return getPreferenceString(EMAIL);
}

private String getPreferenceString(String key) {
    SharedPreferences settings = ctx.getSharedPreferences(PREFS_NAME,
MODE_PRIVATE);
    return settings.getString(key, null);
}
```

Kód 10 – Ukázka práce se sdíleným nastavením

UIBuilder

Obsáhlá třída, která obsahuje metody pro zobrazování různých vyskakovacích oken, bočního menu nebo vyplnění informací o totemu nebo lokaci v aktivitě *ActivityHome*. V dalších odstavcích budou popsány některé z těchto metod.

Nejobsáhlejší metoda této třídy je metoda pro zobrazení okna pro vytvoření a editaci totemu. Do této metody vstupují dva parametry *LocationTotem* a *Context*. Nejprve metoda vytvoří *builder* a následně zjistí, jestli je vyplněn u totemu název. Pokud tomu tak je, jedná se o editaci totemu, a ne o jeho vytvoření. V tomto případě se vyplní všechny údaje o totemu do textových polí. Mezi tím se musí oknu přiřadit rozvržení ze zdrojů, v tomto případě *totem_edit_view.xml* (viz příloha A). Dále definuje metodu, která nastane po kliknutí na ikonu totemu, tedy zobrazí vyskakovací okno pro výběr barvy totemu. Jde o vyskakovací okno *SpectrumDialog*. V poslední řadě se nastavují jednotlivá tlačítka okna ve spodní části, pro zrušení a potvrzení vytvoření, resp. upravení. Tlačítko pro potvrzení obsahuje validaci všech textových polí a vložení totemu do databáze.

2.3.6 ServerFirebase

Tento balíček obsahuje třídy zabývající se komunikací se serverem, konkrétně s Firebase serverem. Obsahuje dvě základní třídy *Database* a *Authentication*.

Authentication

Tato třída se stará o registraci a přihlášení uživatele skrze Firebase. V následujících odstavcích budou popsány dvě základní metody, co jim předchází a co následuje. Budou to metody pro přihlášení a registraci pomocí emailu a hesla. Tyto metody jsou volány z aktivit *ActivityLoginMaster* a *ActivityRegistrateMaster*.

Metoda *loginUser()* očekává tři atributy a to email, heslo a prvek, který se zobrazuje při načítání. Nejdříve metoda zjistí, zda nejsou email a heslo prázdné metodou *isEmpty()* třídy *TextUtils*. Pokud některý z řetězců je prázdný, vypíše chybovou zprávu pomocí zprávy *Toast*. Poté se zavolá metoda *createUserWithEmailAndPassword()* nad instancí třídy *FirebaseAuth* opět s parametry *email* a *password*. Tato metoda vytvoří instanci generické třídy *Task<AuthResult>*, které nastaví metodu, která se zavolá po dokončení přihlášení.

Metoda pracuje obdobně jako metoda pro přihlášení, proto ji nebude dána pozornost. Po obou těchto metodách následuje metoda *onLogin()*, tedy pokud přihlášení dopadne úspěšně. V této metodě se uloží informace o přihlášeném uživateli do sdílených prostředků a přemístí uživatele na aktivitu *ActivityHome* s příznakem, že jde o první spuštění této aktivity.

Database

Třída *Database* se stará o nahrávání dat na server a jejich získání na požádání nebo při jejich změně. Dělí se na dvě základní části, a to část pro uživatele v roli rodiče a dítěte. V následujících odstavcích bude věnována pozornost části pro uživatele v roli dítěte a potom pro uživatele v roli rodiče.

Registrace se skládá ze třech navazujících metod (viz příloha B). Nejdříve se zjistí jednoznačný kód pro komunikaci. Následně se vygeneruje šestimístný hash, skládající se z velkých písmen a číslic. Prověří se jeho unikátnost na serveru. Pokud unikátní není, proces se opakuje. Poté se nahrají informace o zařízení na server pod tímto kódem, tím je zařízení registrováno. Nakonec se tento kód zapíše do textového pole a resetuje se ukazatel dostupnosti kódu.

Obsahuje také metodu na zamknutí, prodloužení dostupnosti kódu, které pouze upraví proměnnou *UNLOCK_TO*. Dále metoda, která reaguje na změny informací o uživateli jako počet je připojených zařízení a čas, kdy byla naposledy nahraná lokace na server. Velmi důležitou metodou je metoda nahrání lokace na server. Tato metoda je ukázána níže.

```
public static void uploadLocation(MyLocation myLocation, Context ctx) {
    String slaveHash = Settings.getInstance(ctx).getSlaveHash();
    FirebaseDatabase database = FirebaseDatabase.getInstance();

    database.getReference(TABLE_SLAVES + "/" + slaveHash + "/" +
LOCATIONS + "/" +
myLocation.getTimestamp()).setValue(myLocation.toMap());

    database.getReference(TABLE_SLAVES + "/" + slaveHash + "/" +
LAST_UPDATE).setValue(System.currentTimeMillis());
}
```

Kód 11 – Metoda pro nahrání lokace

Tato třída obsahuje mnoho metod pro roli rodiče, nejdůležitější jsou však ty metody, které se zavolají po přihlášení uživatele. Výchozí metodou je metoda *onLogin()*. Nejdříve metoda stáhne všechny změněné informace o uživateli týkající se účtu, totemů a připojených dětí. Poté nastaví metody, které se zavolají při změně některých z vybraných dat. Těmto metodám se říká listener v tomto případě jsou to instance třídy *ValueEventListener*. Tato třída obsahuje dvě metody, a to *onDataChange* a *onCancelled*. Instance této třídy se uchovávají ve statických proměnných této třídy, aby mohly být při odhlášení vypnuty.

Data na serveru jsou ukládána ve stromové struktuře v datovém formátu JSON. Struktura databáze ve webové aplikaci Firebase může vypadat jako v obrázku níže. Obsahuje klíčované hodnoty libovolně zapouzdřené do sebe. Klíče musí být ve své vrstvě unikátní, například u lokací je klíčem čas, kdy byla lokace pořízena.

Obrázek 16 – Struktura Firebase databáze

3 ZÁVĚR

Úkolem této bakalářské práce bylo vytvořit mobilní aplikaci na platformu Android, která se zabývá sledováním poloh jiných zařízeních. Výsledkem se stala aplikace „*Watch my baby“*, jež slouží pro sledování lokací dětí. Lokace jsou zobrazovány pouze mezi propojenými zařízeními. Hlavní prvkem je tedy zaznamenávat a zobrazovat aktuální polohu dětí na mapě.

V aktuálním stavu aplikace disponuje pouze omezenými možnostmi a je spousta funkcí, které by ji rozšířili, z důvodu konkurenceschopnosti aplikace. V současném stavu aplikace nedisponuje komunikací mezi uživateli, a proto by mohl být přidán chat mezi uživateli. Pro spolehlivé spojení mezi zařízeními by také mohla být přidána komunikace, skrze SMS zprávy, když zařízení nejsou připojené k internetu. Hlavní a nejdůležitější součástí by měly být přenášené informace, které by měly být co možná nejpřesnější. V tuto chvíli je rychlost pohybu zaznamenávána zařízením, což je velmi nepřesné a z toho důvodu by bylo lepší ji nahradit dopočítanou rychlostí. Tyto informace by mohly být také rozšířeny například o stav baterie.

Dále by mohla být nahrazena metoda ukládání dat, jelikož v současném stavu jsou data uloženy jak v lokální databázi SQLite, tak i v databázi Firebase. Tato duplicita vznikla na začátku vývoje, kdy ještě nebylo pracováno se službou Firebase.

Při vývoji této aplikace jsem získal představu o vývoji aplikací na mobilní platformu. Myslím si, že tento druh aplikace má své využití, zejména pro rodiče, kteří mají starost o své děti.

4 ZDROJE

[1] ARLOW, Jim a Ila NEUSTADT. *UML 2 a unifikovaný proces vývoje aplikací: objektově orientovaná analýza a návrh prakticky*. 2., aktualiz. a dopl. vyd. Brno: Computer Press, 2007. ISBN 978-80-251-1503-9.

[2] Firebase. Firebase [online]. 2017 [cit. 2017-04-26]. Dostupné z: https://firebase.google.com/

[3] Requesting Permissions at Run Time. Developers [online]. 2017 [cit. 2017-04-26]. Dostupné z: https://developer.android.com/training/permissions/requesting.html

[4] ALLEN, Grant. *Android 4: průvodce programováním mobilních aplikací*. Brno: Computer Press, 2013. ISBN 978-80-251-3782-6.

[5] MURACH, Joel. *Murach's Android programming: training & reference*. ISBN 978-1-890774-71-4.

[6] MEDNIEKS, Zigurd R. *Programming Android*. Beijing: O'Reilly, 2012. ISBN 978-1449316648.

[7] LACKO, Ľuboslav. *Vývoj aplikací pro Android*. Brno: Computer Press, 2015. ISBN 978-80-251-4347-6.

[8] Material design guidelines. *Material Design* [online]. 2017 [cit. 2017-04-26]. Dostupné z: https://material.io/guidelines/

5 PŘÍLOHY

Kód v kompletní podobě je dostupný na přiloženém CD.

Příloha A – *Část rozvržení totem_edit_view.xml*

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/contentEditTotem"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context="com.rhsoft.watchmybaby.activities.ActivityMain" >

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="vertical">

        <android.support.design.widget.TextInputLayout
            android:id="@+id/wrapperTitle"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:hint="Title">

            <EditText
                android:id="@+id/edtTotemTitle"
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:inputType="textCapSentences"
                android:maxLines="1"
                android:shadowColor="@android:color/black"
                android:textColor="@android:color/black" />
        </android.support.design.widget.TextInputLayout>

        // Vynechané prvky (radius, time_from, time_to)

        <android.support.design.widget.TextInputLayout
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_gravity="center_horizontal"
            android:gravity="center"
            android:layout_marginLeft="6dp"
            android:layout_marginRight="6dp">

            <TextView
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:textSize="12dp"
                android:text="Choose color"
                android:layout_marginBottom="7dp"/>

            <ImageView
                android:id="@+id/imgViewColorPicker"
                android:layout_width="80dp"
                android:layout_height="80dp"
                android:layout_marginBottom="0dp"
                android:src="@drawable/totem_blue"/>

        </android.support.design.widget.TextInputLayout>
    </LinearLayout>
</RelativeLayout>
```

Příloha B – *Metoda na registraci zařízení dítěte*

```
public static void createSlave(Context ctx) {
    String token = (new InstanceIdService()).getToken();
    checkHashCodeAvailability(token, ctx);
}

private static void checkHashCodeAvailability(final String token, final
Context ctx) {
    final FirebaseDatabase database = FirebaseDatabase.getInstance();
    final String slaveHash = UUID.randomUUID().toString().substring(0,
6).toUpperCase();

    database.getReference(TABLE_SLAVES + "/" + slaveHash + "/" +
CLOUD_MESSAGING_TOKEN).addListenerForSingleValueEvent(new
ValueEventListener() {
        @Override
        public void onDataChange(DataSnapshot dataSnapshot) {
            String value = (String) dataSnapshot.getValue();
            if (value == null) {
                registerSlaveHash(slaveHash, token, ctx);
            } else {
                checkHashCodeAvailability(token, ctx);
            }
        }
        @Override
        public void onCancelled(DatabaseError databaseError) {
        }
    });
}

private static void registerSlaveHash(final String slaveHash, String token,
final Context ctx) {
    FirebaseDatabase database = FirebaseDatabase.getInstance();
    database.getReference(TABLE_SLAVES + "/" + slaveHash + "/" +
CLOUD_MESSAGING_TOKEN).setValue(token);
    database.getReference(TABLE_SLAVES + "/" + slaveHash + "/" +
CREATED).setValue(System.currentTimeMillis());
    database.getReference(TABLE_SLAVES + "/" + slaveHash + "/" +
UNLOCK_TO).setValue(System.currentTimeMillis() + Slave.ACCESS_TIME);
    database.getReference(TABLE_SLAVES + "/" + slaveHash + "/" +
LAST_UPDATE).setValue(System.currentTimeMillis());
    database.getReference(TABLE_SLAVES + "/" + slaveHash + "/" +
CONNECTED_USERS).setValue(0);

    Settings.getInstance(ctx).setSlaveHash(slaveHash);

    new Handler(new Handler.Callback() {
        @Override
        public boolean handleMessage(Message msg) {
            ((TextView) ((Activity)
ctx).findViewById(R.id.textHash)).setText(slaveHash);
            return true;
        }
    }).sendEmptyMessage(0);
    try {
        ((ActivityConnectMaster) ctx).fillProgressBar();
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```